COMPUTER COMMAND AND CONTROL COMPANY

2300 CHESTNUT STREET, SUITE 230 • PHILADELPHIA, PA 19103
215-854-0555 FAX: 215-854-0665

(2)

FINAL REPORT

SYSTEM ENGINEERING AUTOMATION (SEA) FOR DISTRIBUTED SYSTEMS

CONTRACT NO. N00014-91-C-0183
CDRL SEQUENCE NO. A002

AUGUST 1992

Prepared for:

Department of the Navy
Office of Naval Research
Code 1211
Arlington, VA 22217-5000

TABLE OF CONTENTS

LIST OF FIGURES

FINAL REPORT

SYSTEM ENGINEERING AUTOMATION (SEA) FOR DISTRIBUTED SYSTEMS

Abstract

This Final Report describes the work performed in the area of System Engineering Automation (SEA) for Distributed Systems for Contract No. N00014-91-C-0183. The work focussed on researching the exchange of information between design capture and design optimization techniques as part of the DESTINATION project. Design Structuring and Allocation Optimization (DESTINATION) is an ongoing research project at the Naval Surface Warfare Center (NSWC) to provide a new methodology for design optimization and trade off analysis of real-time systems. This project produced a version of the enhanced and extended DESTINATION Interface Specification (DIS).

The need for DIS arises from the inherent adaptiveness of the DESTINATION system to a wide range of source and target tools. DIS not only allows DESTINATION to coexist with various systems, but also dictates standards for a comprehensive way of capturing design information. The basic structure of DIS reflects a method of extracting/incorporating design information that is otherwise not available across a collection of tools. DIS accommodates the identification of additional design information, allowing for customization of the source and target tools. The focus of the DIS research and development work is currently in the area of system logical modelling and implementation modelling.

1. Introduction

One of the primary thrusts behind the Systems Design Synthesis project of the Naval Surface Warfare Center's (NSWC) Engineering of Complex Systems (ECS) Technology Block Research Program is to provide a new methodology for systems engineers in the area of Design Optimization and Trade-Off Analysis. Systems engineers require such a new methodology to cost effectively construct and maintain increasingly complex mission-critical, real-time systems.

Application complexity has increased not only due to functional demands, but also because of technological advances. The present and future combat systems must respond to an expanding theater of commands, as well as the requirement to perform in an integrated manner. Technologically, the advent of parallel computers and high speed networks opens many opportunities to provide greater defense capabilities. These functional and technical factors greatly increase the design space that the systems engineer must explore in search of a design that satisfies all requirements. The idea behind design optimization and trade-off analysis is to provide the systems engineer with the necessary tools and techniques to systematically evaluate and exploit the vast design space.

DESTINATION is the name given to the NSWC research effort that focuses on developing the necessary tools and techniques to support such a methodology for design optimization and trade-off analysis [HoNH]. The emphasis on this project is design structuring and resource allocation tools and techniques. The design structuring involves making decisions regarding decomposition/recomposition and fragmentation/defragmentation of hierarchical designs. Resource allocation includes the mapping of logical design objects onto implementation resources in a near-optimal manner.

The need for an interface specification to perform design optimization first arose on a predecessor project to DESTINATION called EDA or Expert Design Advisor [HoHN]. The first version of the interface specification, developed for EDA, was used to standardize the format of inputs for the development of four resource allocation optimization algorithms—Gantt [Pear], Data-Oriented, Genetic [Davi91] [Gold], and Simulated Annealling [KiGV]. Reuse of the same data structures reduces the size of the development effort and increases the ability to adapt to new algorithms. The need for the exchange of information between various front-end case tools has initiated the development work on standards for the interface specification that would enhance portability, adaptability, maintainability and extensibility for a wide range of source/target tools.

To better understand the function and structure of the DESTINATION Interface Specification, it is necessary to further explain the associated DESTINATION methodology. By reviewing the context diagram in Figure 1, we can see the DESTINATION methodology's scope and contribution within the systems life cycle.

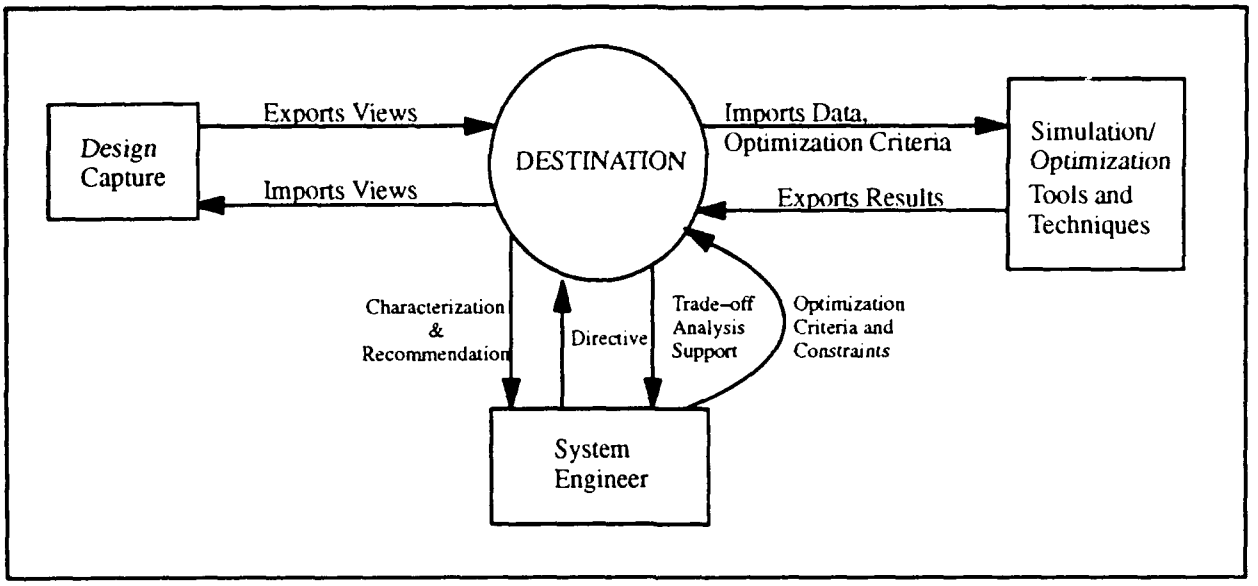

Figure 1: DESTINATION Context Diagram.

The human systems engineer plays a critical role within the methodology. The systems engineer's input is fundamental for selecting the subject of design optimization and evaluation, applying analysis techniques and interpreting results. The methodology supports the systems engineer by making characterizations and recommendations. The systems engineer may accept or override these outputs.

A complete scenario of steps can be mapped into the context diagram to describe the methodology.

1. The systems engineer gives a directive to select a design capture view for analysis.
2. DESTINATION interacts with the system's engineer to determine the following:
    a. System characterization.
    b. Formulation of design goals.
    c. Application of design rules.
    d. Recommendation for use of simulation/optimization tools and techniques.
3. The system's engineer directs DESTINATION to import the necessary data for the selected simulation/optimization tools and techniques.
4. Results from simulation/optimization are exported to DESTINATION for evaluation.
5. When satisfied with the achievement of design goals, any modification to the design capture views are imported to the Design Capture system to maintain consistency. Although these steps are listed sequentially, it is expected that there will be a high degree of iteration within and among these steps, particularly when performing trade-off analysis.

Many approaches have been followed that specify what to capture, such as, structured analysis and design [WaMe], object-oriented design [BOOC], and how to capture it. One of the most robust approaches to be defined, which is consistent with earlier DESTINATION research

efforts, was jointly developed by NAVSWC and Trident Systems Corporation ([Kare], [Hoan]). This approach to forward design capture represents one set of information that may be incorporated into the DESTINATION methodology for analysis. Though DESTINATION is not restricted to any particular Design Capture approach, the NAVSWC/Trident approach is one of the most robust and places the strongest demands on DESTINATION, so it is advantageous to use from a research prospective. Furthermore, use of this approach insures integrated results within the System Design Synthesis project of the ECS block program.

Basically, the forward design capture accepts design information according to three models: the conceptual model, the logical model, and the implementation model. Each is described below.

The conceptual model captures the operational ideas of the system from the perspective of the operational environment and information modelling. The environmental view establishes the conditions and environment in which the system must operate including a description of the system architecture's scope and boundaries, test plan, and operational scenarios. The conceptual model allows the system engineering team and the customer to form a clear understanding of the subject system.

The logical model includes a description of the functional and behavioral views of the system, without regard for any particular implementation decisions. The emphasis within this design capture model is on what the system should do as opposed to how it should do it. The behavioral view provides an understanding of the system from a dynamic perspective.

The implementation model documents the hardware, software and human resources which represent a particular embodiment of the system under design. The hardware architecture describes the physical resources of the system including the components, interconnection topology and protocol, and rationale for selection. The software architecture describes the Computer Software Configuration Items (CSCIs) and the executable software tasks including the messages passed between tasks. The human resource description includes the number of personnel required to operate the system under various conditions and the level of training and experience for each operator.

There is no restriction on what design methodologies may be used within the development of any of the three models.

Likewise, any number of simulation/optimization tools and techniques are available for use within DESTINATION. Optimization algorithms that may be applicable for use include computation/communication-oriented, genetic search and simulated annealing. Simulation techniques that may be interwoven with the optimization algorithms include petri-net simulation (e.g. SES/workbench, ADAS), queueing theory [CCCC], and general purpose simulation languages (e.g. Simscript). Future advances in both optimization and simulation are to be expected.

2. Requirements of the DESTINATION Interface Specification (DIS)

As described in the previous section, the DESTINATION Interface Specification is the layer of data structures and export/import routines that permit application information to flow in and out of DESTINATION. From one perspective, DIS bridges the design capture facilities with optimization decisions and from another perspective, DIS integrates the execution of simulation

models and optimization algorithms with design evaluation and recommendations. This iterative path of capturing, modelling, evaluating and recommending becomes significantly more streamlined by having a consistent and robust medium of exchange.

A number of requirements impacted the development of DIS. These requirements can also be viewed as motivating factors for making the investment in DIS.

1. Tool Independence

   It is desirable to have a methodology be independent of a particular toolset. There may be financial and training constraints that oppose acquiring a toolset, particularly when a comparable one may already be in place. In the context of design capture, for instance, there are several Front-End Computer-Aided Software Engineering (FE-CASE) tools that are operational within Department of Defense (DOD) programs, most notably, Cadre's Teamwork and IDE's Software Through Pictures (StP). The interface to DESTINATION should handle data from Teamwork just as easily as data from StP.

   Certainly, as part of the access routines into the FE-CASE system's repository, there will be some effort that is not reusable. The goal is to minimize this effort. Figure 2 shows how this is done in the context of interfacing with Cadre's Teamwork.

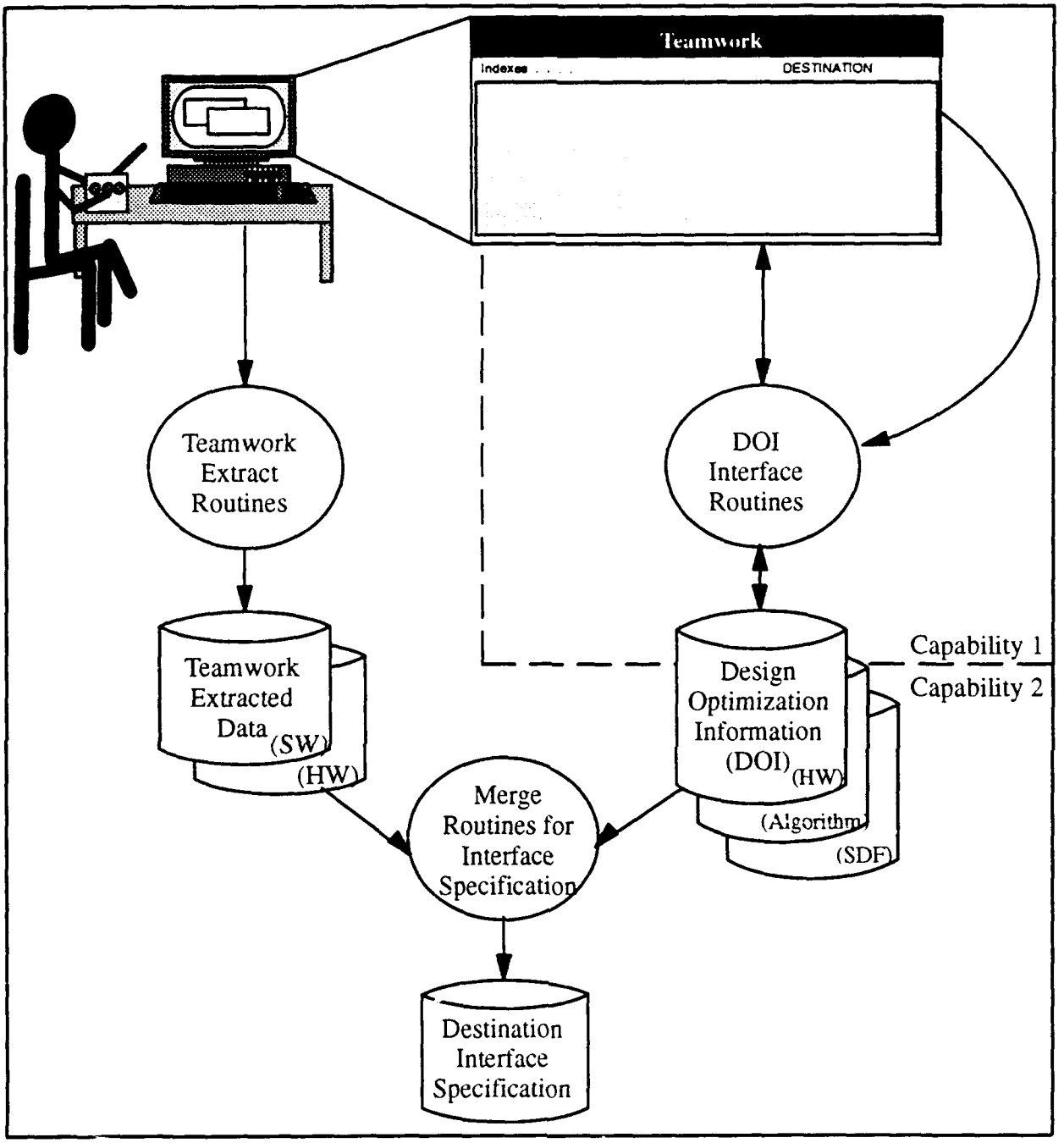

Figure 2: Design Capture Interface.

Two capabilities are shown in Figure 2. The first capability provides the systems engineer with the capability to supplement the design capture process with information for design optimization. There are three types of design optimization information (DOI) that have been included as part of the design capture supplement: System Design Factors [HoNH], data required by optimization algorithms and hardware resource descriptions and characteristics. The second capability extracts information stored directly in the Front-End CASE system's repository representing the information contained within the CASE graphics (bubbles, flows, connections, etc.) The two information sources, the Front-End CASE dependent data and the Front-End CASE independent DOI, are then merged to create a DIS compatible file.

A similar, though possibly more complicated, choice of tools to develop interfaces exists within the Simulation System/Optimization Technique domain as in the FE-CASE area.

2. Implementation Independence
   The DESTINATION toolset contains many interconnected subsystems, such as for design characterization, design evaluation, and for making recommendations regarding design structuring and resource allocation. Development of these subsystems can proceed more independently by sharing the DIS among them.

   Furthermore, developers of algorithms for resource allocation, scheduling, and design structuring can utilize the DIS as a departure point for their innovation. DESTINATION then provides a convenient proving ground for determining the situations where the algorithm performs best. This makes for a win-win situation for DESTINATION and algorithm developers: there will be an increased likelihood that their algorithms will be transferred to practical use and likewise DESTINATION's library of algorithms on which it bases its optimization recommendations will progressively expand. It is expected that the algorithm developers will, in turn, uncover additional requirements for the DIS and through feedback DIS will progressively improve.

3. Supports Incomplete Information
   If optimization is to be performed on a bottom-up basis, there may be substantial information that may not have been provided on a higher level. To proceed under these circumstances, default values can be associated with the DIS data structures and be supplied as required by the optimization algorithms.

4. Gain Wide Acceptance
   DIS must be designed so that it can be used widely, as described in Figure 1, by systems engineers, algorithm developers, tool vendors and standards bodies.

5. Transportability
   DIS should facilitate the transportability of design capture, design optimization and simulation information from one computer environment to another.

6. Uniformity and Cohesiveness
   The DIS model should be simple and uniform, while minimizing the amount of concepts, types and classes of operations.

7. Implementability
   Vendors of simulation systems, front-end CASE systems, and algorithm developers should be able to utilize DIS with only a reasonable effort. The design of DIS should allow for flexibility in implementation while maintaining consistent operational semantics.

8. Extensibility
   As mentioned above, tool vendors, algorithm developers and systems engineers will uncover additional requirements on DIS. DIS should not preclude any extensions to its scope to satisfy evolving needs.

9. Performance
   The DIS design must allow for efficient operation from both external access of design capture and simulation systems as well as from internal DESTINATION procedures.

To satisfy these requirements, several basic design decisions for DIS were made.

1. Represent the data structures for easy mapping onto a flat ASCII file. This accommodates the requirements for acceptability, transportability, implementability, extensibility, and performance.
2. Utilize Ada as the language for formally specifying DIS. There were several underlying reasons for this:
    a. Ada is a DOD standard.
    b. Ada is widely available on many computing platforms.
    c. Ada's package facilities and specification/body separation could be used to express multiple layers of abstraction.
    d. Ada was very successful in its use as a specification language for the Ada Semantic Interface Specification (ASIS) definition [BISp]. ASIS is a vendor-independent, non-proprietary bridge between Ada libraries and Ada tools.

There are a number of alternative methods for specifying the interface. English was dismissed as being two ambiguous. Use of formalisms like the Backus Naur Form (BNF) and Extended Backus Naur Form (EBNF) has the advantage of concise accuracy allowing little room for ambiguities and vagueness but does not allow high level representation. C was not used because of its lack of abstraction facilities. C++ and Common Lisp Object System (CLOS) are viable alternatives, particularly due to their strong object orientation and inheritance facilities. Presently, they lack the standardization and DOD acceptance of Ada. There are systems and associated languages that specialize in interface definition and actually automatically generate some code necessary for declaring and accessing the interface [NEST]. These facilities warrant further investigation, but are not DOD standards.

There are a number of potentially useful integration standards that are emerging, such as PCIS, Case Integration Services (CIS), IRDS, CASE Document Interface Format (CDIF), IEEE-P1175 and NGCR's PSSWG [StSh]. None of these efforts, however, are directly working in the area of design optimization and lack representation of much needed information. Through planned participation with these working groups and standards bodies DIS should beneficially impact these efforts.

3. Components of the Interface Specification

3.1. Overview

The current version of DIS, 2.0, is divided into several packages at its top level. Each of these packages is comprised of lower level packages. This basic structure, at present, is shown in Figure 3.

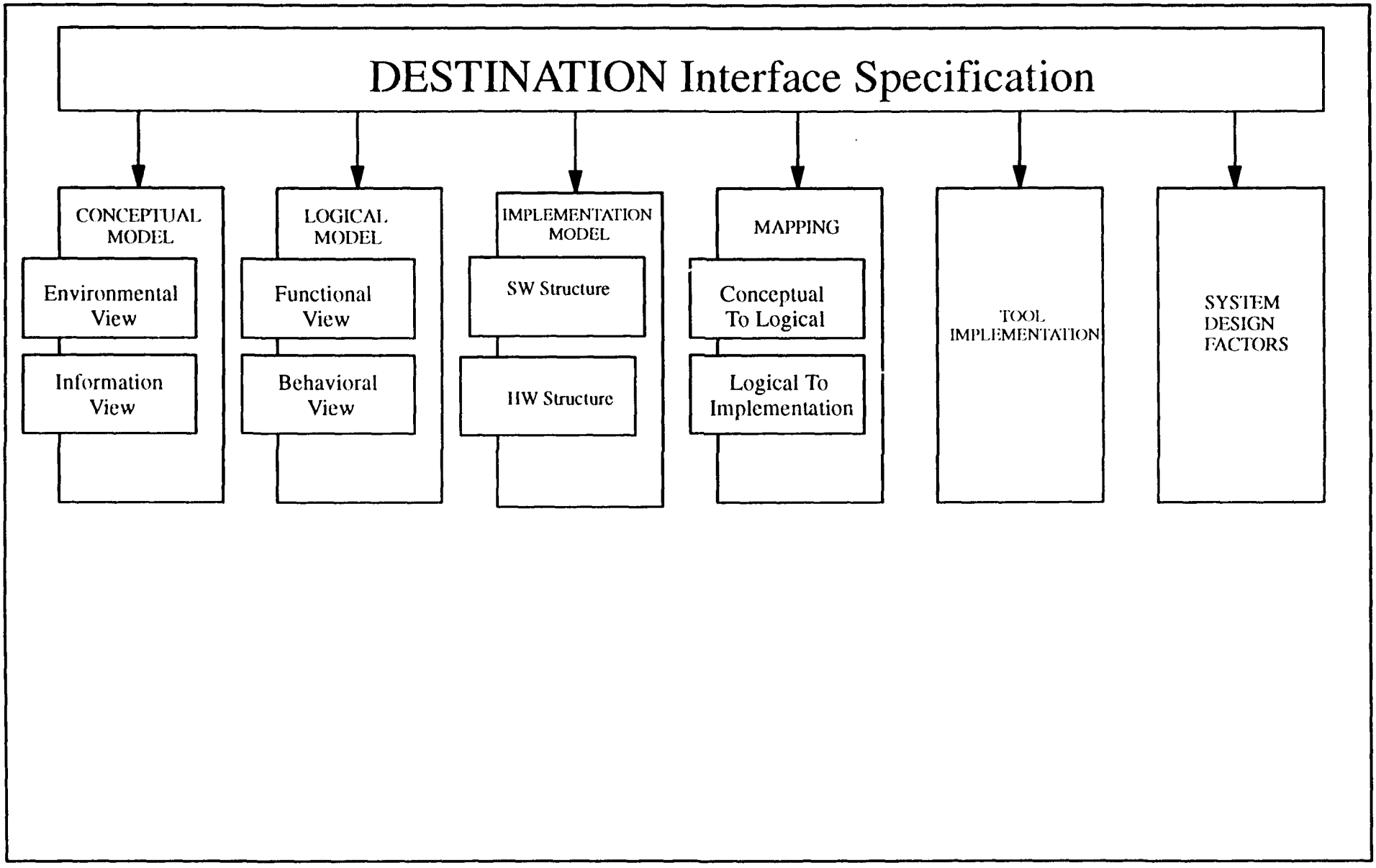

Figure 3: DESTINATION Interface Specification.

The following sections describe the logical and implementation models, as well as system design factor in greater detail with brief descriptions of the structural components and the Ada data type declarations for some of the major components. In the last section, the Teamwork and TAE+ screens, the supporting Ada routines and the DIS formats will be described in steps. This is where the emphasis of the current effort has been. Development of a technical report describing the other packages is in progress.

3.2. Logical Model

The logical model describes the functional and behavioral views of the system. The emphasis within this design capture model is on what the system should do as opposed to how it should do it.

The logical model contains information representing functional decomposition of system and interactions between the decomposed functions of the system through the functional view, and the dynamic operations of the decomposed functions at a different time under different situations and conditions through the behavioral view of the system.

Representations are needed for the following information:

1. The flow graph to depict the candidate system in the static decomposition configuration.
2. The control graph to depict the candidate system in the dynamic decomposition configuration.
3. Design factors derived from requirements. Design factors describe the non-functional aspects and the critical real-time information of the system, such as fault-tolerance, reliability, accuracy, quality, security, deadlines and reconfiguration.

The logical representation of the system consists of one or more logical views, i.e., a list of the logical views.

Ada type and C++ class declarations for the DESTINATION Interface Specification can be found in Appendix A and B.

Each logical view consists of a functional view and a behavioral view. The functional view contains a data flow diagram in which a flow object represents the decomposed functions of the system and a flow edge represents the interactions between two decomposed functions of the system.

The behavioral view contains a state transition diagram and a process activation table. The state transition diagram describes all the existing states of the system and possible transitions between states under different conditions. The process activation table describes under which condition each process is activated.

To better explain the data structures represented within the logical model (and also later for the implementation model), graphic figures have been provided. Figure 4 contains a legend of the graphical notations. In Figures 5, 6, 7, and 8, there are three types of edges connecting the data structures representing the logical model:

1. An *is linked to* relation showing that the data structure is part of a linked list. For each of these relations there are two pointers—one for the next occurrence in the list and one for the previous occurrence in the list. This double linked list structure allows for reduced programming in traversing the list.

2. A *parent/child* relation to indicate that a structure decomposes into another data structure. The decomposition implies that when a data structure contains another data structure, the former data structure may be viewed as the parent and the latter is viewed as the child.

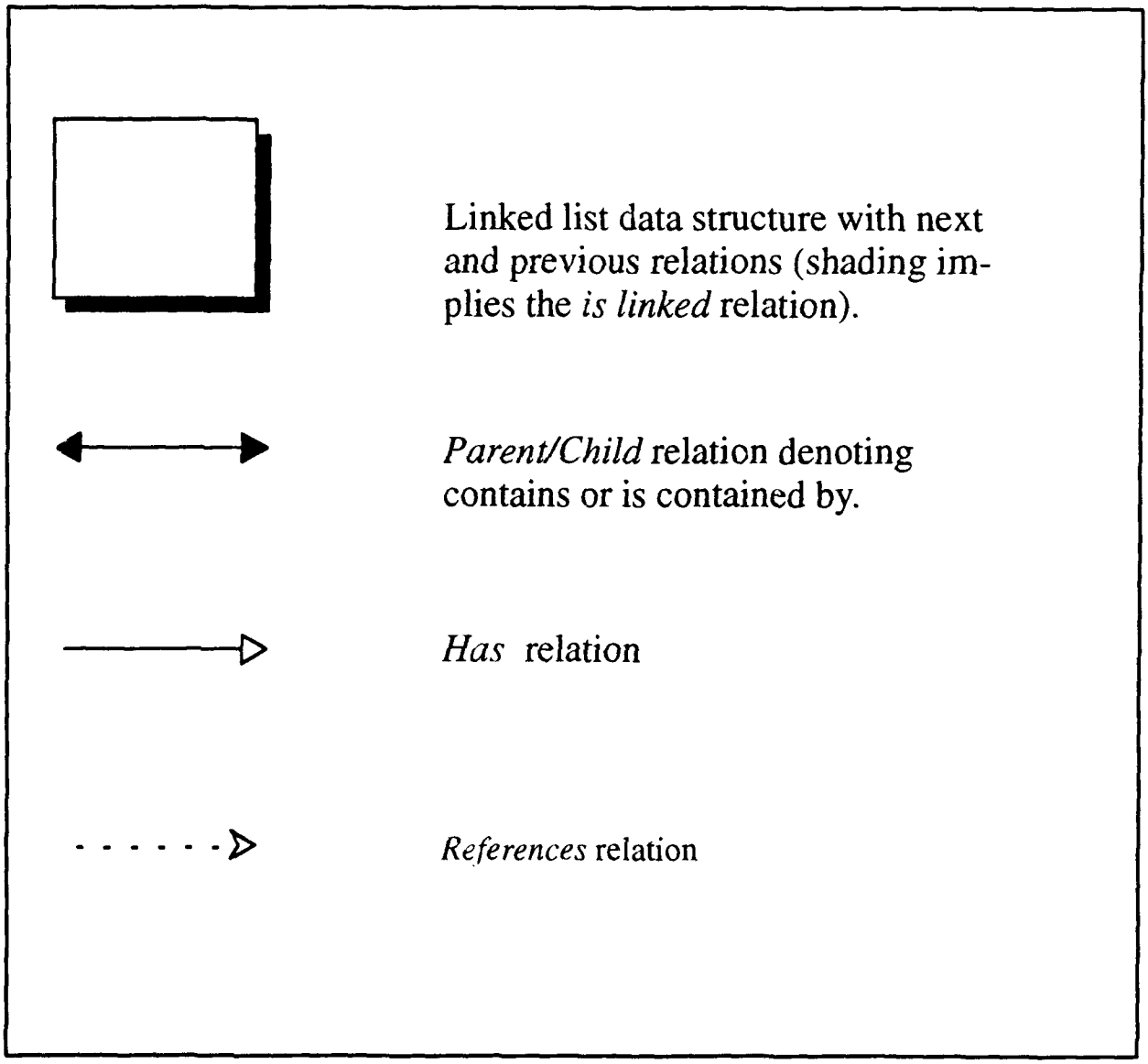

Figure 4: Nodes and Edges Used to Describe DIS Components.

3. A *has* relation to denote a logical pairing with other data structures.

4. A *references* relation when two structures share a common data element.

3.2.1. The Functional View

The functional view describes the system structure that captures how functions in the system are decomposed and how they interact with each other. Similar to the same design capture view using Yourdon–DeMarco structured analysis method in [Hoan], this functional view represents flow object and data flow between them in a graphical and hierarchical way as shown in Figure 5. The graphical and hierarchical view helps the system engineers to analyze the functional structure of the system.

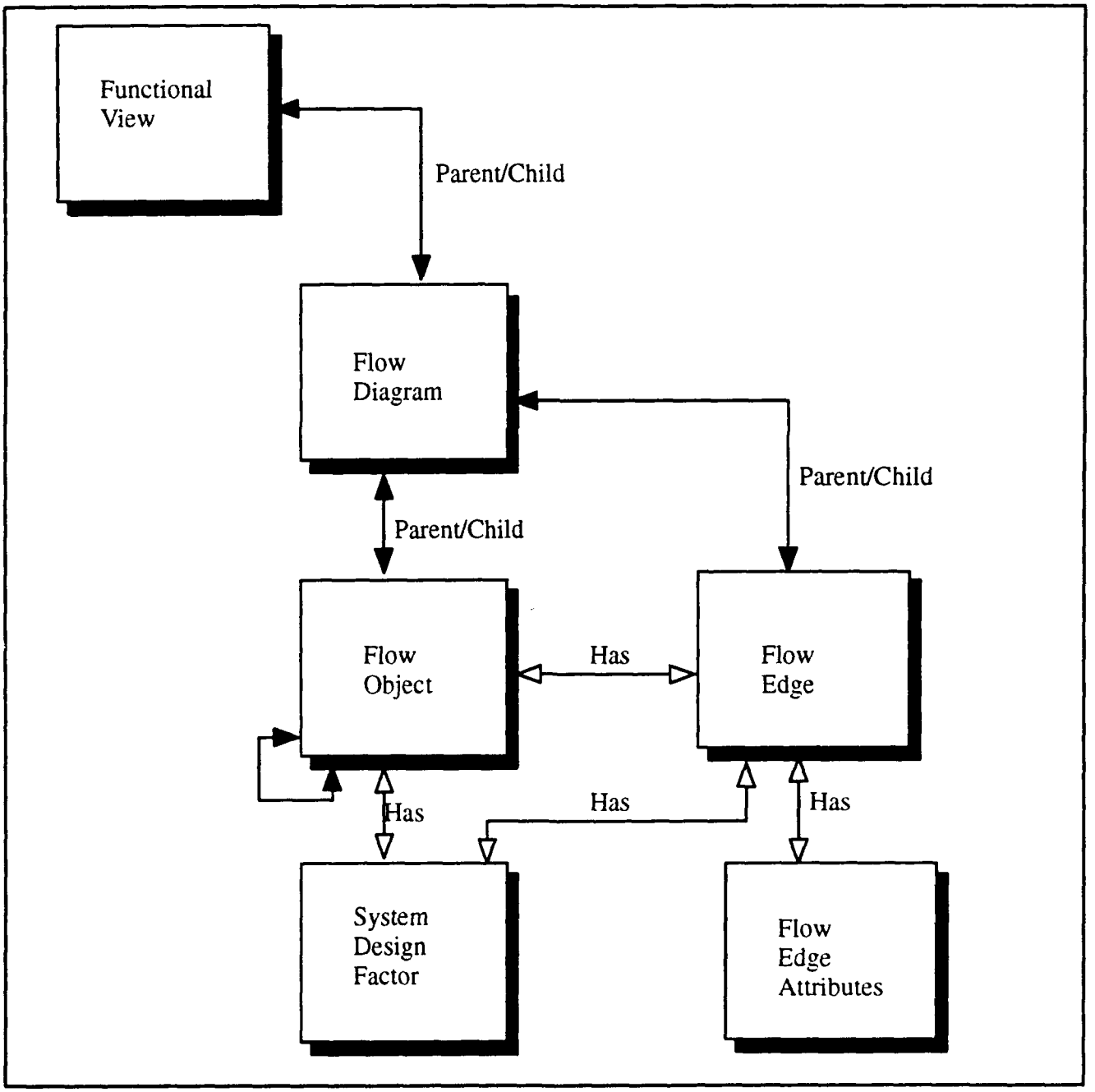

Figure 5: Functional View Architecture.

Additionally, this view specifies the non-functional aspects of the system with System Design Factor. Briefly, the System Design Factor (SDF) is one of the mechanisms that provides system engineers with the capabilities to:

1. Specify various design goals and criteria
2. Quantify various aspects of the design
3. Perform trade-offs among different design goals through optimizations.[NgHo]

Defining such System Design Factors can overcome the inadequacies found in conventional design capture views, in which the non-functional properties of the system are not supported.

Figure 5 shows the architecture of the functional view. The data types for this view is in Appendix A and B. As shown in the figure and the code, each functional view contains its own flow diagram. This flow diagram is represented by flow objects and flow edges between these objects. A flow object represents a decomposed function of the system and contains the flow object information as follows:

1. The hierarchical, sibling and nesting relations between flow objects.
2. The nested flow object list including their flow edges.
3. The flow object description, characterization and design factors.

A flow edge represents the relation between decomposed functions of the system or the flow objects in the flow diagram. Attributes of a flow edge contain the following information:

1. The hierarchical (sibling) relations between objects.
2. The direction of the flow under the predefined conditions.
3. Design factor information, such as frequency, duration, unit, accuracy, etc.

3.2.2. The Behavioral View.

The behavioral view describes the dynamic behavior of the system under control. This view captures the operations of the system at different times under different situations and conditions. Similar to the functional view of the system, the behavioral view represents the states of the system and their transitions as well as the process activations in a graphical and hierarchical way, such that the system engineers can analyze the behavioral construction of the system. Additionally, the view captures the non-functional and critical real-time information of the system, such as deadline and reconfiguration. Defining some design factors in this view overcomes some problems stated for the same design capture view in [Hoan], in which this non-functional and critical real-time information is not specified.

Figure 6 shows the architecture of the behavioral view of the system. The data type for this view may be found in Appendix A (Ada) and B (C++). Each behavioral view is represented by the list of the state transition diagram and the process activation table. Each state transition diagram is a directed graph, in which nodes represent the states of the system and edges represent state transitions under the different conditions. The process activation shows the dynamic behavior of the system at different times under different situations and conditions. Note

the distinction between functions and processes in this logical model, such that the function implies the static computation task of the system and the process implies the dynamic operation of the computational task.

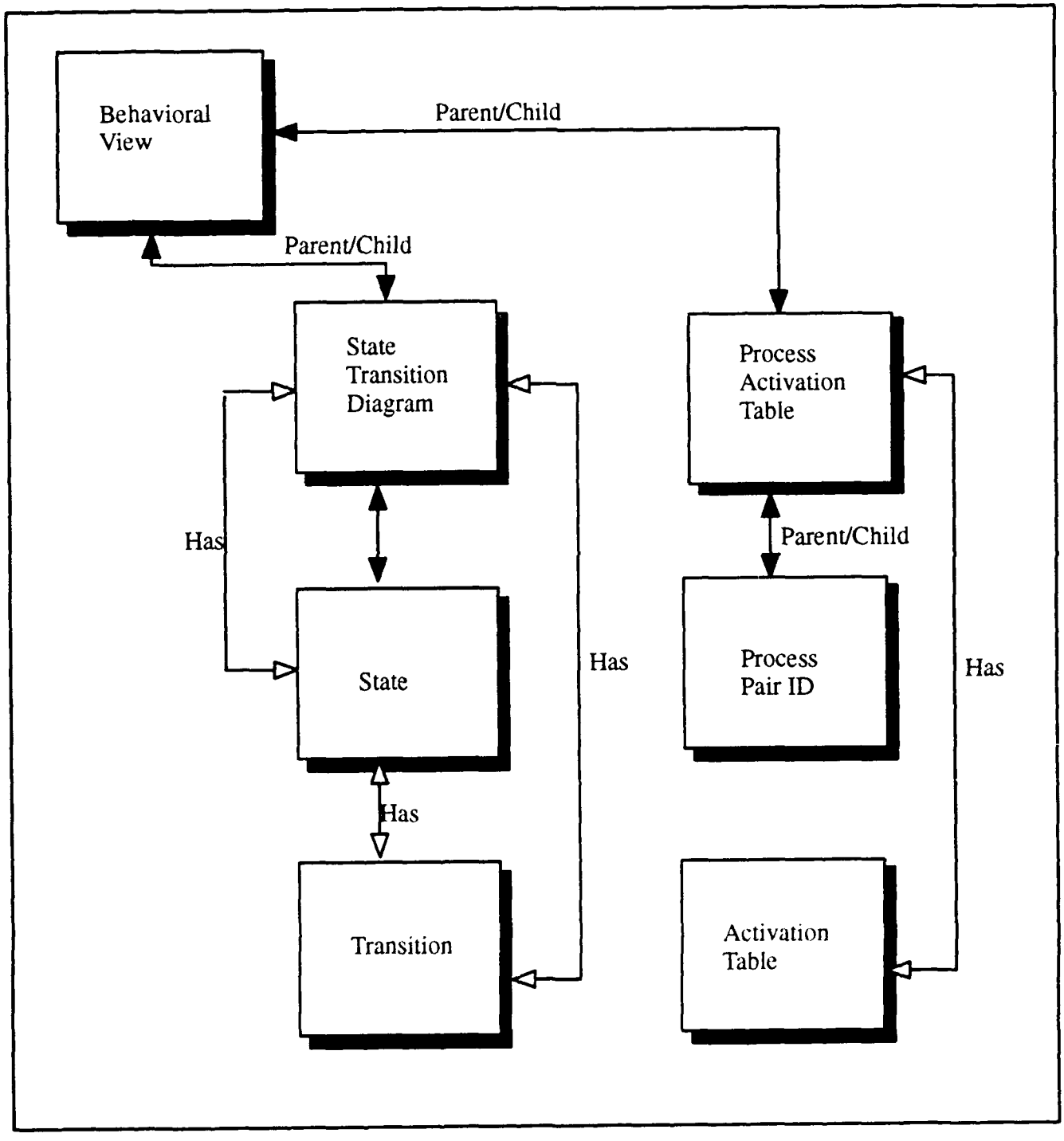

Figure 6: Behavioral View Architecture.

3.3. Implementation Model

The implementation model contains data representations for building the system and for making and analyzing decisions for resource allocation and design structuring. Representations are needed for the following information:

1. A task graph to depict the candidate software configurations.
2. A resource graph to depict the candidate hardware configurations.
3. Constraints derived from requirements. Constraints are presently divided into two categories: placement constraints and timing constraints. These constraints impact the effectiveness of optimization. Each one of these types of constraints contain several sub-types of constraints that will be described further below.

The implementation model of a real-time system consists of one or more implementation views (i.e., a list of implementation views). Each implementation view consists of a software structure diagram, a hardware structure diagram and a list of mappings of software components to hardware resources. The type declaration for this information in the DIS is contained with the DIS_implementation_view_type declarations in Appendices C and D.

The term *structure diagram* is used to reference a collection of directed graphs, drawn with respect to a selected methodology, that captures information about a set of components and their relations along with any hierarchical decomposition. For example, a tree of data flow diagrams may be considered as one type of *structure diagram.*

The list of mappings is provided, since for the same software and hardware structure diagrams, we may apply different allocation tools and techniques to determine possible mappings. Each of the components of the implementation view are further described below.

3.3.1. Software Structure

Figure 7 represents the software structure architecture. The data types for the software structure is shown in Appendix C (in Ada) and Appendix D (in C++). Each software structure diagram is represented by a list of modules and a list of edges between modules. A *module* represents a collection of nodes and edges within a *structure diagram* (as explained above). A *module* could be considered as a level of decomposition.

Software modules can be nested and each module includes its own task graph. Task graphs cannot be nested since the node of a task graph cannot be a module. However, nested relations between tasks can be captured using nested modules. Our view is that the task represents a separately executable computational entity.

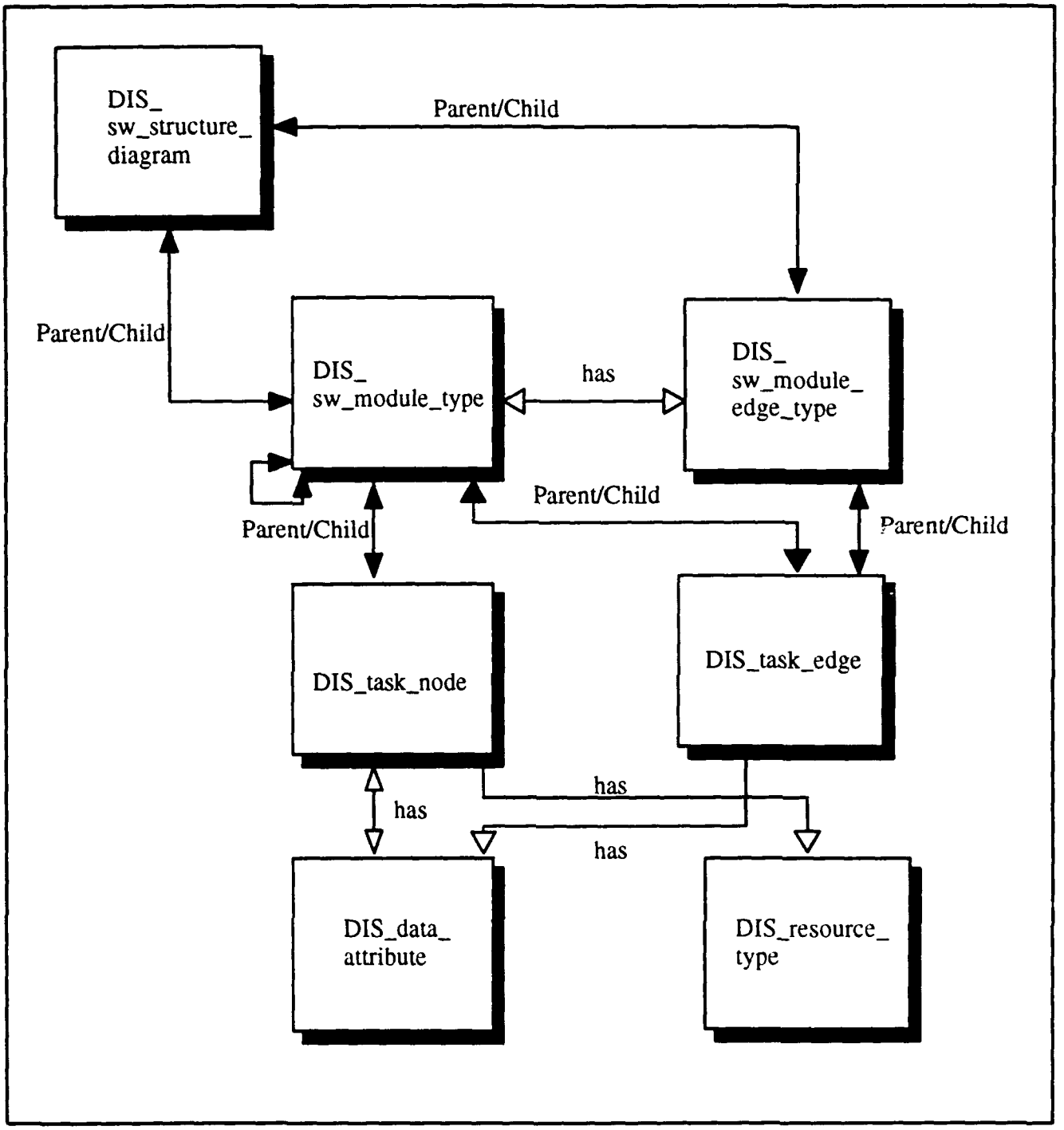

Figure 7: Software Structure Architecture.

A software module contains the following information:

1. The hierarchical, sibling and nesting relations between modules.
2. The identity of task graphs that belong to the module. In addition, there are two special kinds of edges (called entry_super_edge and exit_super_edge). They are used to identify the entry and exit points of the task graph at the module level.

A task graph is a directed graph: each node denotes a schedulable computational entity and an edge represents a precedence relation between two nodes. For each task node in the *DIS_task_node* structure, there is a *task_input_list* to identify input data and *task_output_list* to identify output data generated by the task. In addition, *task_predecessor_list* identifies tasks that execute before the task and *task_successor_list* identifies tasks that execute after the task. There is an *and_or* flag associated with the above four task lists that specifies whether all input (or output) data are needed (or generated) by the task. This information is required by some optimization algorithms. Each task may include timing information such as ready time, deadline and duration. In addition, it identifies resources it needs. For resource needs, *DIS_resource_type* identifies the resource a task needs and the amount it needs. For each task edge, *task_edge_data* identifies the data associated with the edge along with the duration of availability of the data. In addition, *from_task_node* and *to_task_node* specifies the source and destination of the edge.

The declarations for the task node data structure and the task edge data structure are shown in DIS_task_node and DIS_task_edge parts of Appendix C.

3.3.2. Hardware Structure

A hardware structure diagram defines a hardware configuration. A hardware configuration is viewed as consisting of hardware nodes, connected by hardware links. Each node is recursively viewed as consisting of internal nodes that are connected by internal links. The architecture of the hardware structure diagram is shown in Figure 8.

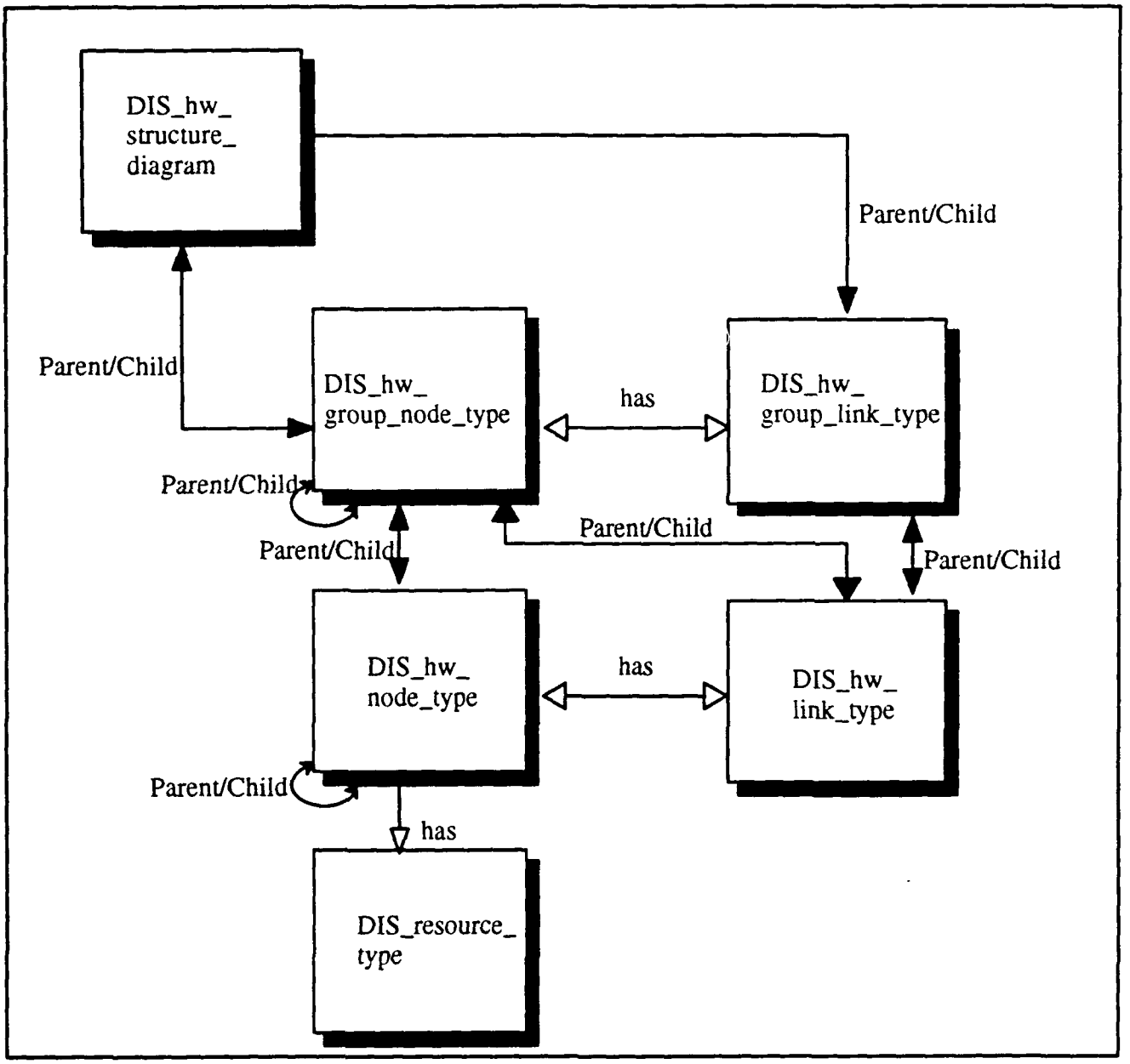

Figure 8: Hardware Structure Architecture.

3.3.3. Mapping Structure

A mapping assignment consists of mapping constraints and task assignments. Figure 9 represents the mapping structure architecture. There are two types of mapping constraints: timing constraints and placement constraints. Each mapping constraint includes a preference value that specifies the importance of meeting the mapping constraint; the magnitude of the value reflects its importance.

The data structure representing the mapping constraints is shown in the DIS_mapping_constraint in Appendix C (in Ada) and Appendix D (in C++). There are four kinds of timing constraints that may be defined for a set of tasks:

1. *complete_within t* means that the set of tasks should complete within *t* time units of each other.
2. *start_within t* means that the set of tasks should start within *t* time units of each other.
3. *complete_path_within t* means that the sequence of the set of tasks should complete within *t* time units from the beginning of the sequence.
4. *complete_start_within t* for two tasks, A and B, means that B should start within *t* after the completion of A.

There are three kinds of placement constraints, each placement constraint is defined on a set of tasks:

1. *place_together* means that the set of tasks should be assigned to the same hardware component.
2. *place_separate* means that the set of tasks should be assigned to different hardware component.
3. *place_at* means that the set of tasks should be assigned at a particular hardware component.

A task assignment is the result of running an allocation algorithm on a set of software and hardware nodes with a set of timing and placement constraints.

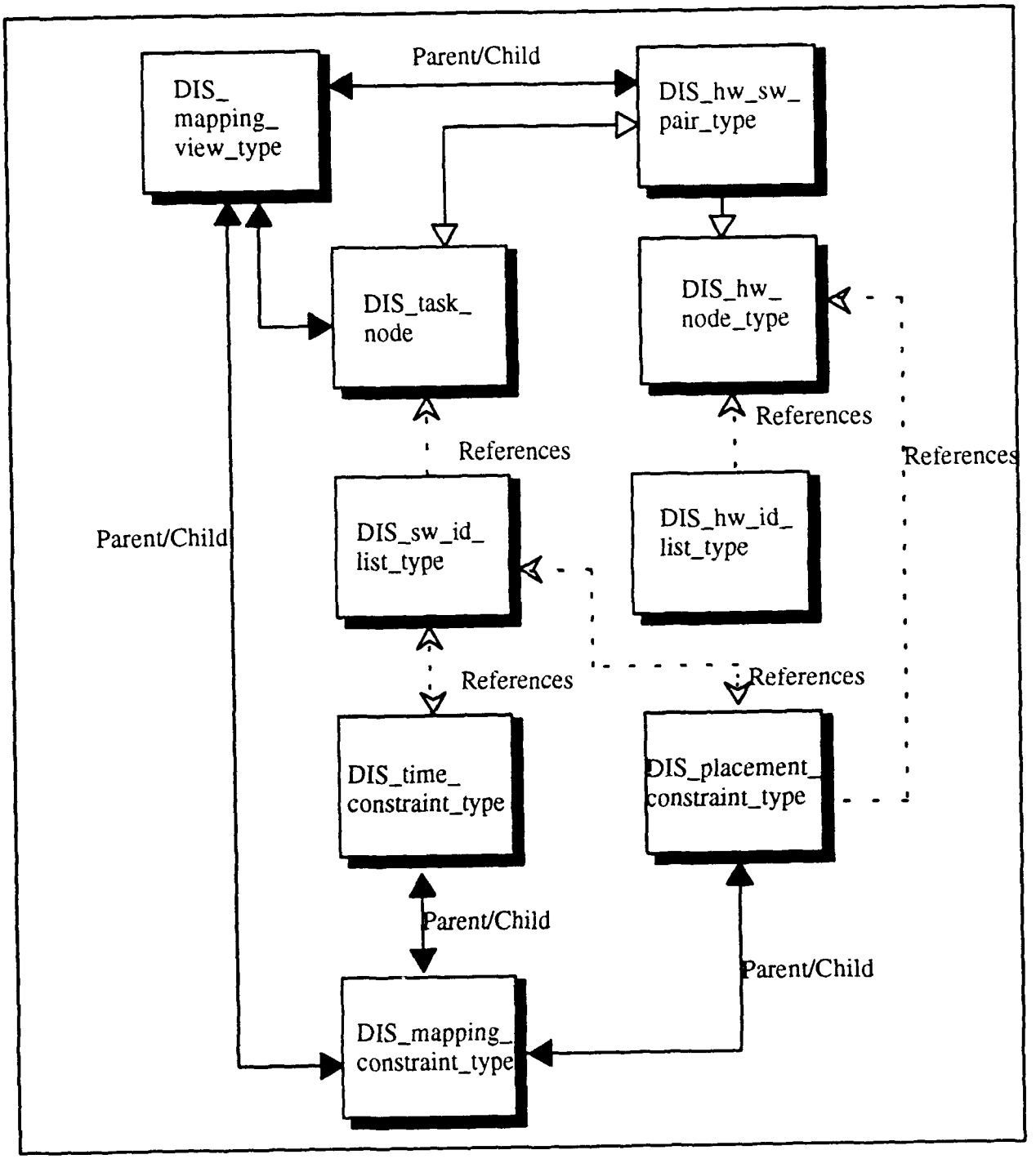

Figure 9: Mapping Structure Architecture.

3.4. System Design Factors

To meet requirement and desired measure of effectiveness in designing a real-time, large and complex system, the system engineers and analysts must be able to specify design goals and criteria, quantify various nonfunctional aspects of the design, and perform trade-offs among different design goals with optimizations.

One of the mechanisms that provide the system engineers with these capabilities is the System Design Factor [NgHo]. This System Design Factor (SDF) describes non-functional aspects of the system, such as performance, dependability, security and real-time responsiveness that all system engineers and analysts should consider to produce the effective system. With the SDF, system engineers and analysts can specify the properties, attributes and characteristics of the system to satisfy all requirements.

In DIS, SDF's can be specified for the logical or implementation model.

In the logical model, SDFs are defined for each flow object and edge in the functional view, and each state transition table and process activation table in the behavioral view. In the impiementation model, SDFs are defined for each component of both software and hardware architectures.

The data types for a SDF is illustrated in Appendices E (in Ada) and Appendix F (in C++).

3.5 Screens, Supporting Routines and File Formats

In this section, the DIS supporting routines and file formats are described. Figure 10 shows the detailed Design Capture Interface for DIS from Figure 2. DIS provides the system engineers with the following two capabilities:

1. The capability to supplement the design capture process with information for design optimization. There are three types of design optimization information:
   (a) System Design Factor.
   (b) Data required by optimization algorithms, such as for timing and placement constraints.
   (c) Hardware resource descriptions and characteristics.
2. The capability to extract information stored directly in the Front-End CASE system's repository representing the information contained within the CASE graphics (bubbles, flows, connects, etc.).

This section describes how a prototype set of desired capabilities was developed for this project. As shown in Figure 10, the above two capabilities are accomplished by extending features of Teamwork by TAE+ routines and Ada action routines. TAE+ is a tool for developing graphical user interfaces. Additional Ada programs are added to perform the actions requested by the user. Cadre's Teamwork was used as a representative CASE tool, however, DIS is not restricted to use of Teamwork. By using the Teamwork Integrated Programming Support Environment (IPSE) and TAE+-generated Ada routines for the graphical interface, the main menu and the data flow diagram entry menus can be extended as shown in Figures 11 and 12.

The Import/Export entries in the DeStinAtiOn of the main menu provides the system engineers with the ability to extract data from Teamwork. (Note that the import capability has not been developed yet.) This is done as follows:

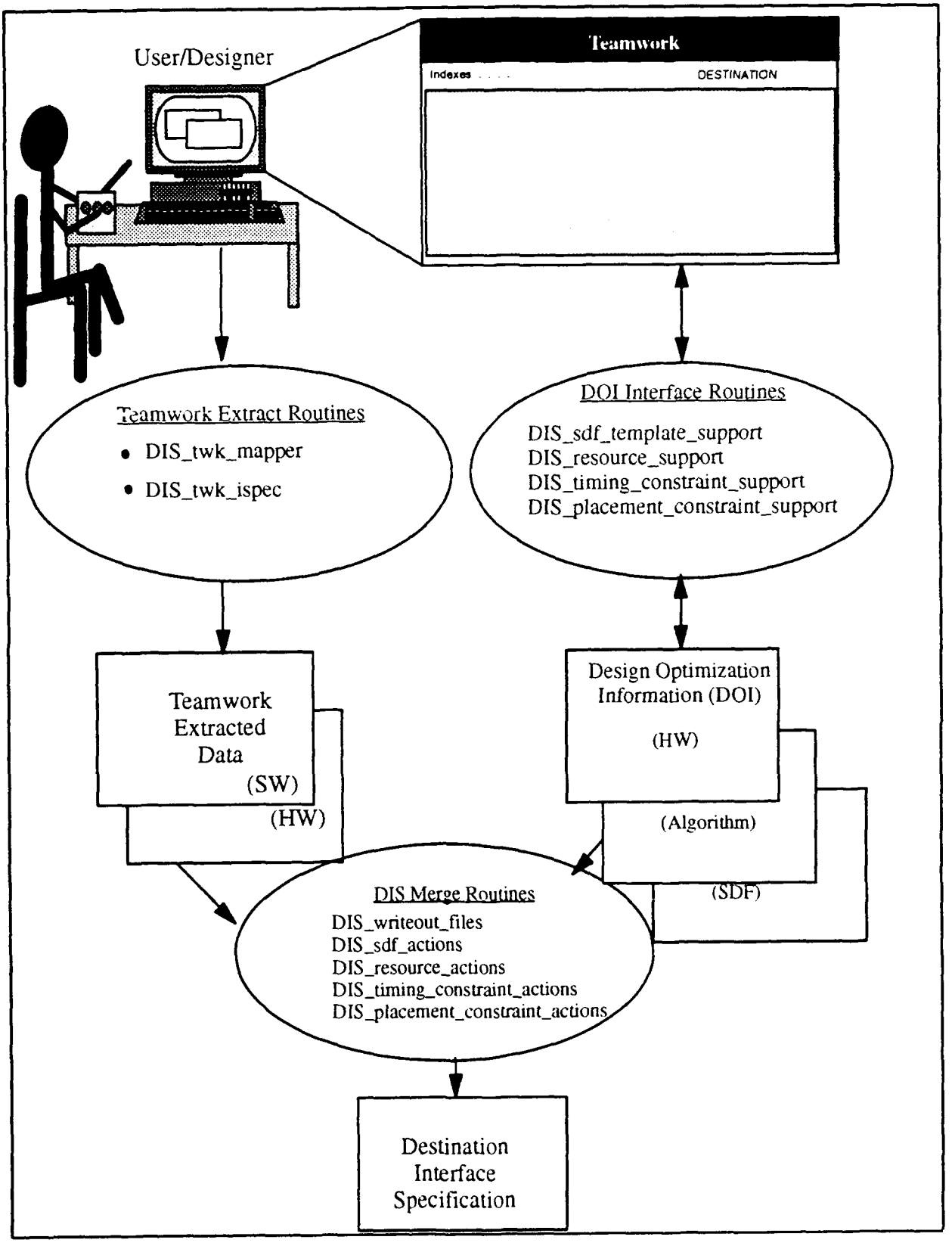

Figure 10: Supporting Routines for Design Capture Interface.

1. When these entries are selected and the model name of the desired configuration is specified, such as for a particular software structure configura-

tion, the Teamwork Extract Routines, as shown in Figure 10 are called—namely, DIS_twk_mapper and DIS_twk_spec.

2. When the Teamwork Extract Routines are called, they extract informations stored in the Teamwork repository, transform them into DIS structures and store the results internally. The Teamwork Extract Routines call the DIS Merge Routines within DIS_writeout_files.

3. Finally, when the DIS Merge Routines for DIS_writeout_files, is called, it gets the internal DIS structure of the given Model configuration, traverses through its hierarchical structure in DIS, classifies each respective information for the predefined component of the DIS record structure and writes these records into their respective files.

The list of the files produced by this DIS_writeout_file routine is as follows:

1. DIS_sw_structure_file for DIS_sw_structure_diagram
2. DIS_sw_module_fil for DIS_sw_module
3. DIS_sw_module_edge_file for DIS_sw_module_edge
4. DIS_sw_task_fil for DIS_sw_task
5. DIS_sw_task_edge for DIS_sw_task_edge
6. DIS_sw_data_attribute_file for DIS_sw_data_attribute.
7. DIS_hw_structure_file for DIS_hw_structure_diagram
8. DIS_hw_group_node_file for DIS_hw_group_node
9. DIS_hw_group_link_file for DIS_hw_group_link
10. DIS_hw_node_file for DIS_hw_node
11. DIS_hw_link_file for DIS_hw_node

As shown in Figure 12, the SDF/Timing_Constraint/Placement_Constraint/ Resource DeStinAtiOn options in the DFD menu provides the system engineers with the first capability described above. This is accomplished as follows:

1. When any of the entries in the DFD pulldown menu titled DeStinAtiOn is selected and any component of the diagram is selected (for example, a task node of software structure configuration), the Design Optimization Information Interface Routines are called as in Figure 10—namely, DIS_sdf_template_support for SDF entry, DIS_resource_support for Resource entry, DIS_timing_constraint_support for Timing Constraint Entry, and DIS_placement_support_constraint for Placement Constraint. As the names imply, the DIS_sdf_template_support routine is to supplement System Design Capture with System Design Factor information, DIS_resource_support routine with hardware resource descriptions and characteristics, and DIS_timing_constraint_support and DIS_placement_constraint_support routines with data required by optimization algorithms.

2. When one of the DOI Interface Routines is called, it displays its respective screens as shown in Figures 13, 14, 15 and 16 so that the system engineers can supplement its respective information. After all information is entered, the respective routine extracts the information from the

screen, represents it in the DIS structure, and calls the respective supporting Ada action routine as a part of the DIS Merger Routines as shown in Figure 10—namely DIS_sdf_actions, DIS_resource_actions, DIS_timing_constraint_actions and DIS_placement_constraint_actions.

3. Finally, as part of the functionality of the Merge Routines, the internal DIS structure is converted into the respective DIS file structure and written out.

The list of the files produced by each routine is as follows:

1. DIS_sdf_file for DIS_sdf
2. DIS_resource_file for DIS_resource
3. DIS_timing_constraint_file for DIS_timing_constraint
4. DIS_placement_constraint_file for DIS_placement_constraint

The Teamwork Extract Routines, DOI Interface Routines, and Merge Routines for DIS are listed in Appendix G. Due to their size, only the specifications of these routines are provided. In Appendix H, the DIS file formats are shown.

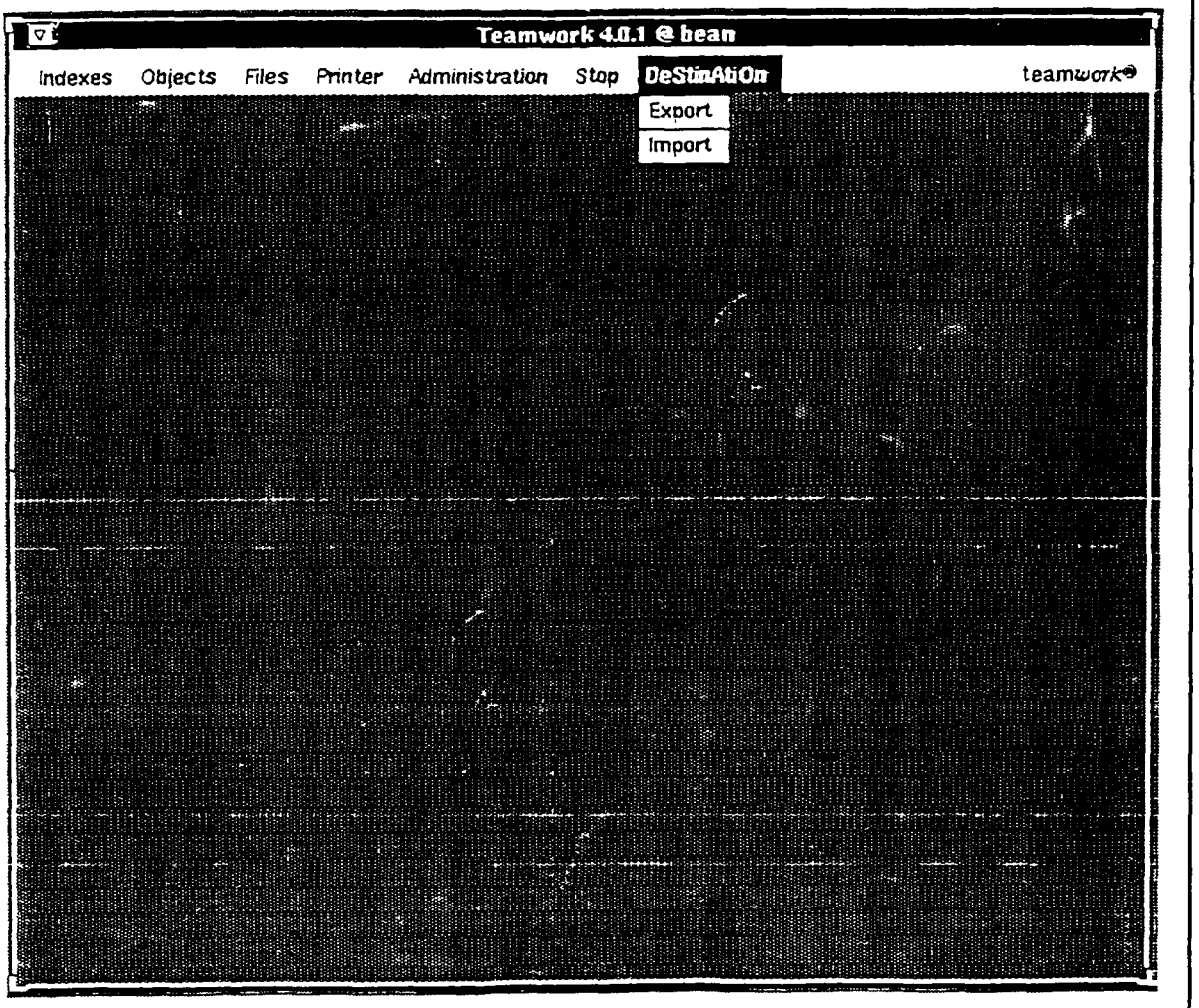

Figure 11: Extended Teamwork Main Menu Options for DESTINATION.

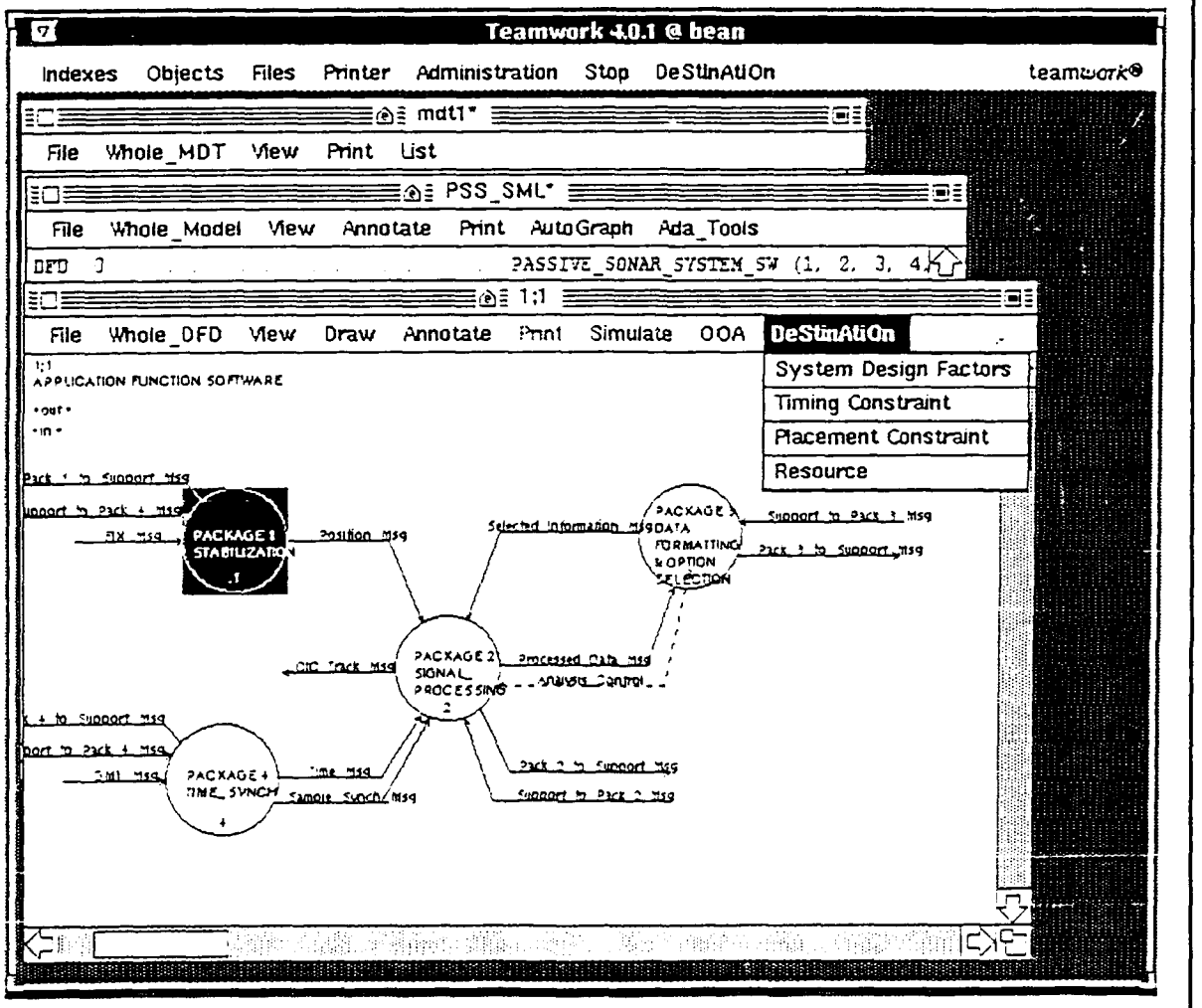

Figure 12: Extended Teamwork Data Flow Diagram Menu Options for DESTINATION.

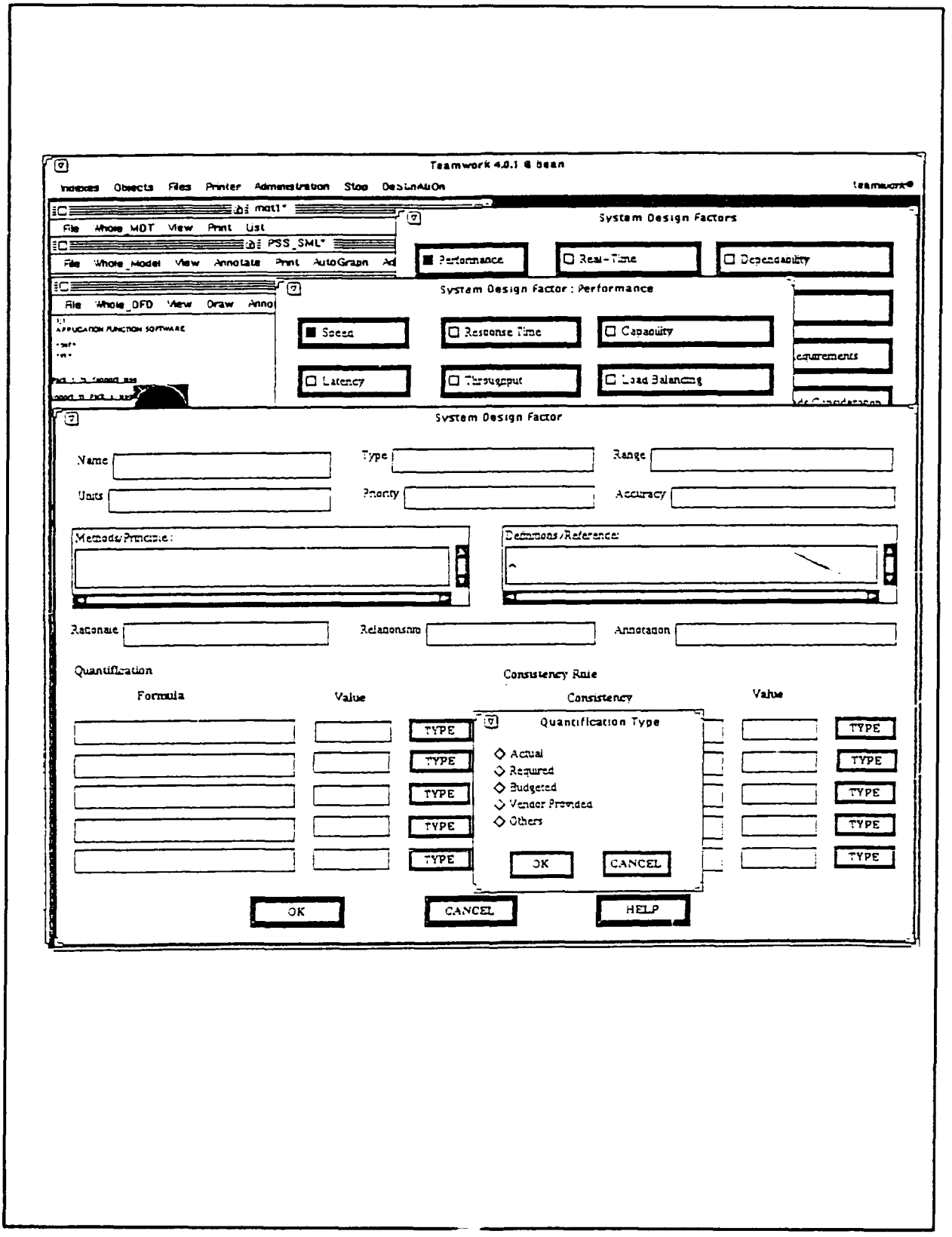

Figure 13: TAE+ Screen for System Design Factor Template.

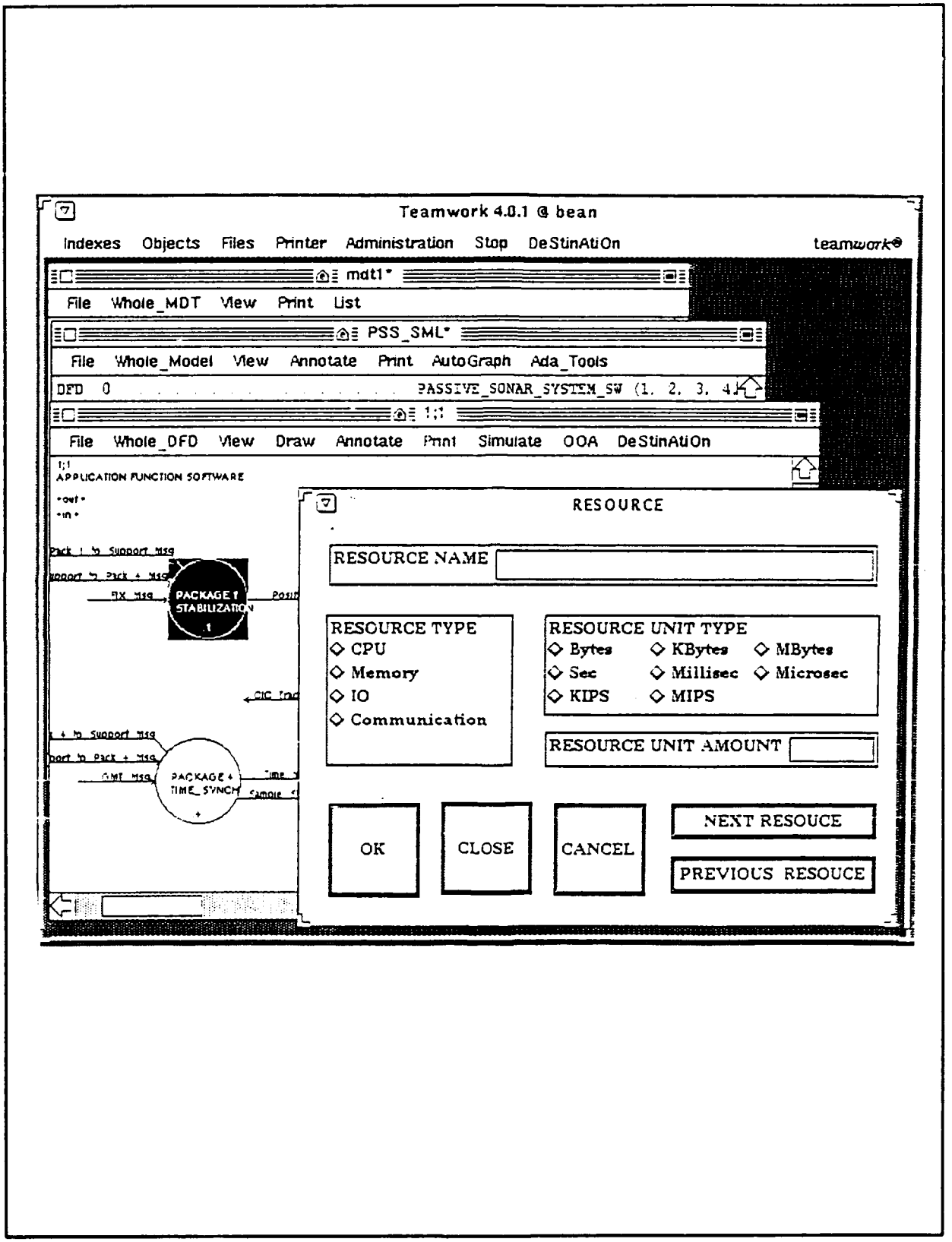

Figure 14: TAE+ Screen for Hardware Resource Description.

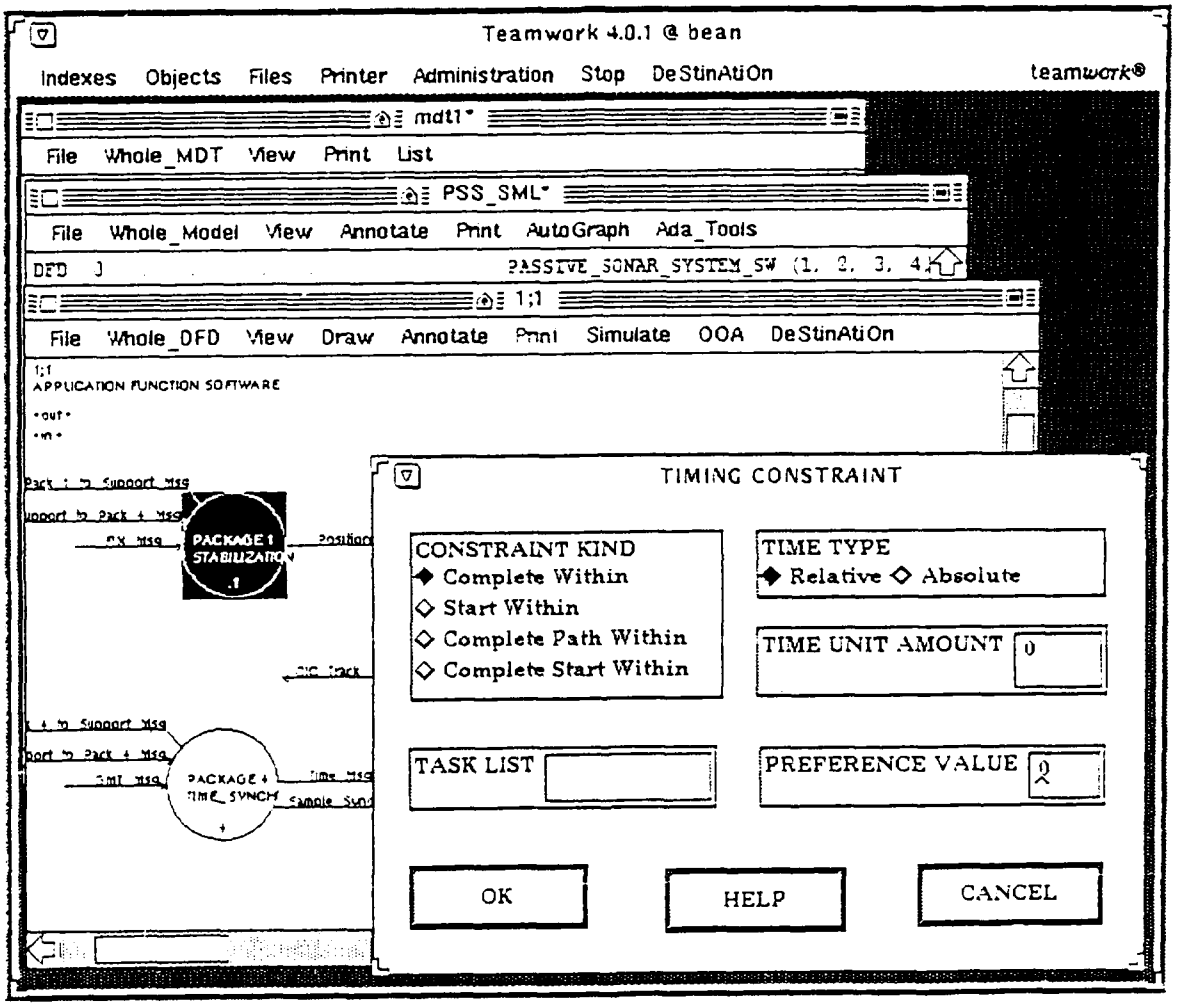

Figure 15: TAE+ Screen for Timing Constraint.

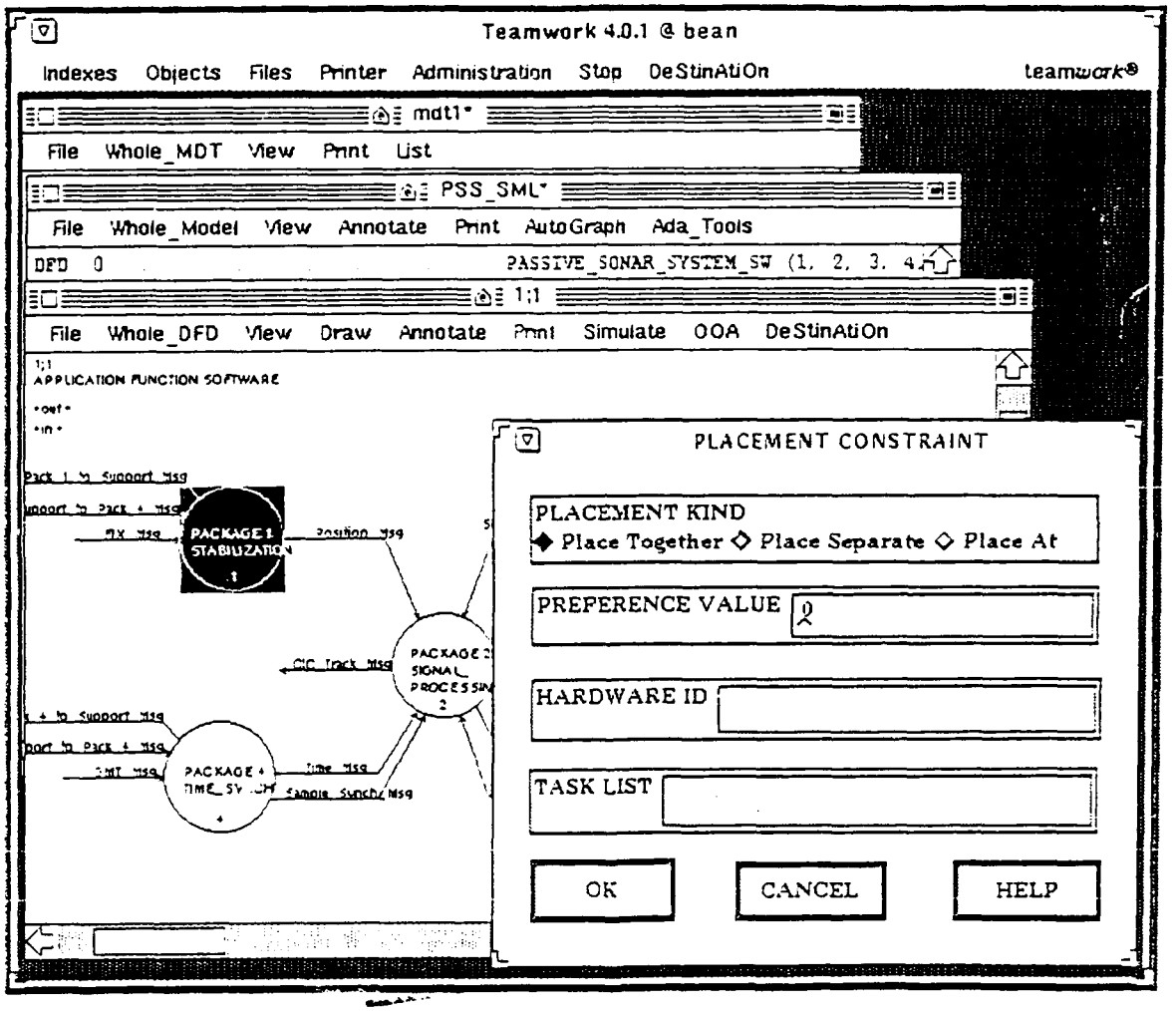

Figure 16: TAE+ Screen for Placement Constraint.

4. Future Directions and Conclusions

The development of DIS is in the first year of an on going effort. There are several tasks that are planned for future development. These are elaborated below.

1. Library of Services
   In addition to setting up standards for data structures, DIS should also provide operations such as load/unload, add, delete, update and queries.

2. Importing Results To Design Capture
   Provide feedback functions to incorporate the results of the optimization back into the design without modifying the structural components.

3. Exporting Results From Simulation/Optimization Systems
   Interface specifications components can represent results from simulation/ optimization systems. This should not only enhance the process, but also provide a standard data representation for developing extract functions for the target systems.

4. Alternative Representation Languages
   Use of high level object-oriented languages to represent DIS has the advantage of inheritance, allowing a higher degree of reusability.

Additionally, now that several iterations of DIS have been produced and, as it becomes more robust, participation in various standards organizations will begin. Involvement in the CASE Document Interchange Format Working Group is expected to begin by third quarter of 1992.

5. References

[BlSp] Bladen, J.B., D. Spenhoff, "Ada Semantic Interface Specification (ASIS)", Proceedings of Tri-Ada '91, pp. 6-15, October 1991.

[BOOC] Booch, G., "Object Oriented Design with Applications", The Benjamin Cummings Publishing Company, Redwood City, CA, 1991.

[CCCC] Computer Command and Control Company, "Software Engineering Environment for Parallel/Distributed Systems", Final Report, Contract #N6092189-C0127, October, 1990.

[Davi] Davis, L. Ed., Handbook of Genetic Algorithms, Van Nostrand Reinhold, 1991.

[Gold] Goldberg, D.E., Genetic Algorithms in Search Optimization and Machine Learning, Addison Wesley, 1989.

[Hoan] Hoang, N.D., "The Essential Views Of Systems Development", Proc. 1st Systems Design Synthesis Technology Workshop, September, 1991.

[HoHN] Howell, S., Hwang, P., Nguyen, C., "Expert Design Advisor", Proc. 5th Jerusalem Conference On Information Technology, IEEE Computer Society Press, Los Alamitos, CA, October 1990, pp. 743-756.

[HoNH] Howell, S., C. Nguyen and P. Hwang, "Design Structuring and Allocation Optimization", Proc. Hawaii International Real-Time Systems Conference, January 1992

[Kare] Karengelen, N., "Multi-Domain Real-Time System Design, Capture and Analysis", Proc. 1st System Design Synthesis Technology Workshop, September 1991.

[KiGV] Kirkpatrick, S., C.D. Gelatt, and M.P. Vecchi, "Optimization by Simulated Annealing", Science, May 13, 1983, Volume 220, No. 4598.

[MoMW] Molini, J.J., S.K. Miason and P.H. Watson, "Real-Time System Scenarios", Proc. 11th Real-Time Systems Symposium, IEEE Computer Society Press, Los Alamitos, CA, December 1990, pp. 214-225.

[NEST] Nestor, J. R., Newcomer, J. M., Giannini, P. and Stone, D. L., "IDL: The Language and Its Implementation", Prentice Hall, 1990.

[NgHo] Nugyen, Cuong M. and Steve L. Howell, System Design Factors, 1992 Complex Systems Engineering Synthesis and Assessment Technology Workshop Report, NSWC, Silver Spring, MD, pp. 147-154.

[Pear] Pearl, J., Heuristics: Intelligent Search Strategies For Computer Problem Solving, Addison-Wesley Publishing Company, Inc., Reading, MA, 1984.

[StSh] Staples, G., D. Shavon, "Earthquake Insurance: One Integration Approach", Software Magazine, pp 41-44, February 1992.

[WaMe] Ward, P., and S. Mellor, Structured Design, Yourdon Press, Englewood Cliffs, NJ, 1972, 2nd ed.